## Teaching Python to management accounting students

Schoute, Martijn

***published in***
The Accounting Educators' Journal
2019

***document version***
Publisher's PDF, also known as Version of record



***citation for published version (APA)***
Schoute, M. (2019). Teaching Python to management accounting students: an illustration using support department cost-allocation methods. *The Accounting Educators' Journal*, *29*(1), 137-161. http://www.aejournal.com/ojs/index.php/aej/article/view/565

*THE ACCOUNTING EDUCATORS' JOURNAL*
**Volume XXIX**
**2019**
**pp. 137-161**

# Teaching Python to Management Accounting Students: An Illustration Using Support Department Cost-Allocation Methods

**Martijn Schoute**
Vrije Universiteit Amsterdam

## Abstract

This paper explores how the Python programming language can be taught to management accounting students using domain-specific examples and exercises. Building on an existing case, the paper presents a number of Python codes that can be used as teaching materials in a management accounting course, and discusses how the case and the Python codes can be used in such a course. The materials cover a topic, support department cost-allocation methods, that is discussed in almost every management accounting course, and also include a relatively new approach known as the lattice allocation method. This topic was mainly chosen because the available methods for allocating the costs of support departments to other departments vary in terms of ease of use, which translates into Python codes that also vary in terms of difficulty and required functionalities.

Keywords: Python programming language, Teaching materials, Support department cost-allocation methods, Lattice allocation method, Python codes.

Acknowledgments: The author would like to acknowledge the helpful comments on earlier versions of the paper offered by Henri Dekker, Malte Max and Eelke Wiersma.

## Introduction

Organizations are increasingly confronted with major new developments in information and communication technology (ICT), ranging from new ICT applications in the execution of operational tasks and processes to the development of new products and services based on real-time user information. These developments, especially those related to data analytics and Big Data, have a big impact on the roles, tasks and responsibilities of accountants[1]. Not only because the structure and processes of organizations - i.e., the focal objects of accountants' measurement and assurance activities - are changing (Lucas et al., 2013), but also because many of the traditional tasks performed by accountants increasingly become automated (Frey & Osborne, 2017). Some scholars suggest that as a consequence, accounting jobs may become obsolete and that the accounting profession may even face extinction (e.g., Frey & Osborne, 2017). Others are less pessimistic and argue that "data analytics and Big Data will instead change task structures within the accounting profession, and this will provide opportunities for accountants to leverage their existing skills in conjunction with newly acquired ones" (Richins et al., 2017, p. 64; see also Lawson, 2018; Lawson & Smith, 2018).[2] These scholars typically argue that in order for accountants to add value in a world of Big Data analytics, they must acquire adequate skills in data analytics,

[1] In this paper, the term 'accountant' is defined broadly and also includes auditors, management accountants (or controllers) and accounting consultants.
[2] See Pincus et al. (2017; but see also Fogarty, 2018) for an interesting, related debate on how technology forces, in addition to financial forces, are dramatically changing the milieu of higher education in the U.S., and on the implications of these forces for change specifically for accounting academia.

as well as (at least) a basic understanding of programming languages such as Python and/or R, as these languages are very useful for data analytics purposes.[3]

In response to these developments, universities and other educational institutes are currently massively examining how to infuse data analytics, often in combination with a programming language, into their accounting curriculum.[4] Dzuranin et al. (2018) have studied how this can best be done, where they distinguish between three approaches: the focused, integrated and hybrid approaches. The focused and integrated approaches respectively entail either including a separate course concentrating on data analytics (and possibly programming) in the program or not including such a separate course but instead having all courses of the curriculum pay at least some attention to it. The hybrid approach, which Dzuranin et al. (2018) find to work best in practice, combines these elements by both including a separate course concentrating on data analytics (and possibly programming), and having all courses of the curriculum pay at least some attention to it. Similarly, building on research that has shown that reinforcing and integrating general skills in a domain-specific context enhances students' learning of these skills (e.g., Bransford et al., 1986), Ballou et al. (2018) also argue that data analytics should be integrated in all courses, and with traditional technical accounting knowledge within these courses, throughout the accounting curriculum.

If one decides to use the hybrid approach (or otherwise the integrated approach) to infuse data analytics and programming into the accounting curriculum, then what is needed is materials that can be used to teach the chosen programming language, such as Python, to accounting students. Although it is possible (and to some extent necessary) to teach these students the basics of the programming language using universal examples, it would be more relevant and effective if this is also done using domain-specific (i.e., accounting) examples and exercises. This paper provides such materials covering a topic that is discussed in almost every management accounting course, namely support department cost-allocation methods, and discusses how these materials can be used. The reasons for choosing this topic are on the one hand, the fact that it is included in almost every management and cost accounting textbook, but also because the available methods for allocating the costs of support departments to other departments vary in terms of ease of use, which translates into Python codes that also vary in terms of difficulty and required functionalities.

The objective of this paper is to explore how the Python programming language can be taught to management accounting students using domain-specific examples and exercises. Building on an existing case, the paper presents a number of Python codes that can be used as teaching materials in a management accounting course, and discusses how the case and the Python codes can be used in such a course. Ideally, these materials are used in either a basic or intermediate management accounting course, when students already have some background knowledge of Python.[5] The topic of support department cost-allocation methods is now typically discussed in

[3] It should be noted that there is some disagreement about how important it is for accounting students to acquire programming skills. For example, in a recent survey among 267 accounting faculty by Dzuranin et al. (2018), respondents rank 'other programming languages (for example R, visual basic)' as the least important software tool for student exposure. Similarly, in a recent survey among 342 accounting professionals by Brink and Stoel (2019), respondents rank 'code writing' as the least important of a number of general skill areas, and 'programming (R, Java, Python)' as the least important of a number of specific application-oriented skills. As argued by Dzuranin et al. (2018, p. 34), these results could "be a result of the emerging nature of the accounting profession's use of these tools or the lack of authoritative guidance requiring the use of certain tools in the field." Moreover, the results of Brink and Stoel (2019) also show that their respondents expect that statistical skills such as 'machine learning' and data-oriented skills such as 'data integration/gathering' and 'data mining', which are typical skills for which (at least) a basic understanding of programming languages is very useful, will become more important in the near future.

[4] As emphasized by Dzuranin et al. (2018, p. 24-25; see also AACSB (2013)), this is "especially true for accounting programs that have separate AACSB [Association to Advance Collegiate Schools of Business] accreditation, given that Accreditation Standard A7: Information Technology Skills and Knowledge for Accounting Graduates requires universities with separate accounting accreditation to include content and learning objectives associated with data analytics and information technology skills in its curriculum."

[5] Alternatively, these materials can also be used in a basic Python course for accounting students, when these students already have some background knowledge of management accounting.

such courses either only conceptually or by using a case where the calculations are done using Excel software.[6] For this paper, the Python programming language[7] is chosen because it is a relatively user-friendly, high-level, multi-purpose, open source[8] programming language that is used in a wide range of domains and technical fields, and that is very useful for the purposes at hand. Also, for many accounting purposes using Python (and related programming languages) has some clear advantages over Excel (such as greater flexibility, the ability to handle bigger datasets, and better reproducibility due to the fact that coding automatically provides an audit trail), which make it valuable for accounting students to acquire (at least) a basic understanding of such languages. It is assumed that the students do not only have Python, but also the NumPy[9] library (for example, as part of the Anaconda[10] package) available on their computer.

The remainder of this paper is structured as follows. First, section 2 briefly discusses the topic of support department cost-allocation methods, including a relatively new approach known as the lattice allocation method. Section 3 discusses a case, taken from Togo (2012), which can be used to teach students how to write and/or adapt Python codes to apply the different methods. Section 4 presents and discusses the Python codes and related exercises. Section 5 provides a discussion on how the case and the Python codes can be used. Finally, section 6 provides a summary and some concluding remarks.

## Support Department Cost-Allocation Methods

Departments within an organization can be divided into two broad categories: operating (or production) departments and support (or service) departments. Operating departments (such as machining and assembly departments) directly produce or distribute an organization's output; support departments (such as human resources and information systems departments) provide services and support to operating departments as well as other support departments. Services provided between support departments are known as interdepartmental or reciprocal services. When such interdepartmental services occur, the allocation of the costs of support departments can become complicated.

There are three widely used methods for allocating the costs of support departments to other departments: the direct, step-down and reciprocal methods. These methods vary in terms of ease of use and accuracy because of how they approach the issue of interdepartmental services. The direct method fully ignores the interdepartmental services and allocates all costs of the support departments directly to the operating departments. This method is relatively simple to use but may result in relatively inaccurate cost allocations, given that there is no recognition of support services provided to other support departments. The step-down (or sequential) method partly recognizes the interdepartmental services by allocating the costs of the support departments to other support departments and the operating departments sequentially in a stepwise fashion, where once the costs of a certain support department are allocated, none are allocated back in subsequent steps. A major drawback of this method is that cost allocations vary depending upon the criteria used for support departments' closures (e.g., based on the costs of each support department or the level of services provided to other support departments). Finally, the reciprocal method fully recognizes the interdepartmental services by allowing reallocations back to each support department. It is the most accurate method, but also the most complicated. Ultimately, when determining which cost-allocation method to use, organizations should consider the extent of interdepartmental services and the cost and benefits associated with each method (Drury, 2018).

Several approaches can be used to apply the reciprocal method. One approach, which is the approach typically illustrated in management and cost accounting textbooks (e.g., Bhimani et al., 2019), uses algebraic substitutions to solve a set of simultaneous equations. Another approach, which students and financial managers typically find to be quite challenging (e.g., Bent & Caplan, 2017; Christensen & Schneider, 2017), uses matrix algebra

[6] See, for example, Christensen and Schneider (2017), Keller (2005, 2015), Stinson (2002) and Togo (2010, 2012) for discussions of how different functionalities of Excel can be used to do the calculations according to the reciprocal method.
[7] See https://docs.python.org/3/
[8] Note that this also makes it interesting for smaller non-research-intensive universities with limited resources (cf. Pelzer & DeLaurell, 2018) to make teaching Python part of their accounting curriculum.
[9] See https://www.numpy.org/
[10] See https://www.anaconda.com/

operations to solve this set of simultaneous equations. Bent and Caplan (2017) recently introduced a new method for allocating costs, which they refer to as the lattice allocation method. Similar to the reciprocal method, this method also uses matrix algebra operations, but without the complexity of solving a set of simultaneous equations. As a consequence, the lattice allocation method is easier to use than the reciprocal method and more accurate than the direct and step-down methods.[11] Moreover, it provides the managers of the operating departments relatively better information about the origins of the costs allocated to them (Bent & Caplan, 2017).

The central technique of lattice allocation is "to multiply [the lattice allocation matrix (a matrix of fractions reflecting the percentages of service provisions among the different departments)] by itself, until the transformed matrix converges to a stable solution and final rules emerge. In other words, one takes this matrix to a sufficiently high power to obtain the desired degree of accuracy for the organization's costing purposes" (Bent & Caplan, 2017, p. 100). The lattice allocation matrix "can be partitioned into quadrants. The […] cells in the upper-left quadrant represent services provided by each [support] department to the other [support department(s)]. The […] cells in the upper-right quadrant represent services provided by each [support] department to the [operating] departments. The […] cells in the lower quadrants reflect the fact that no costs are allocated out of the [operating] departments" (Bent & Caplan, 2017, p. 100). The lattice allocation method builds on the fact that multiplying this matrix by itself, reduces the total amounts in the upper-left quadrant, increases the amounts allocated in the upper-right quadrant, and causes no changes in the lower quadrants, while each row continues to sum to one (Bent & Caplan, 2017). As emphasized by Bent and Caplan (2017), these characteristics are always true with the lattice allocation method. The allocated costs using this method iteratively converge to the results for the reciprocal method, and greater accuracy can be obtained by taking the lattice allocation matrix to a higher power.

## The Togo (2012) Case

Management and cost accounting textbooks typically present a relatively simple case involving only two support departments and two (or three) operating departments, when discussing the topic of support department cost-allocation methods (e.g., Bhimani et al., 2019; Drury, 2018; Seal et al., 2019). The following case is based on data taken from Togo (2012), and is more complicated but also more realistic than such simple cases. The case is about Early Childhood Learning, a manufacturer of electronic educational products for young children. This company has four support departments (human resources (HR), product development (PD), information technology (IT) and maintenance (MA)) and three operating departments (laptop computers (LT), video games (VG) and early readers (ER)). Early Childhood Learning has always used the direct method to allocate the costs of the support departments to the operating departments. However, because it experiences disadvantages of using this method, expects an increase in the number and costs of its support departments, and expects more disproportionality in the usage of their services by planned new versus current products, Early Childhood Learning is currently reconsidering its support department cost-allocation method. The support departments currently allocate costs to operating departments based on the following cost drivers (i.e., cost allocation bases): HR number of employees, PD engineering hours, IT service hours, and MA square footage occupied.

Table 1 is based on Togo (2012) and presents the costs for the four support departments and three operating departments before any cost allocations. In addition, the table also presents the proportion of support services provided by the support departments to the other departments. The -1.00 listed for support departments represents their allocated reciprocated services. Since the total of reciprocated services provided by a support department is equal to 1.00 and all of it will be allocated to other departments, the total for each support department is equal to 0.00. The 1.00 listed for operating departments reflects that these departments allocate 100% of their services to themselves. It is important to emphasize that this case assumes that all costs are variable and that the support departments provide no services to themselves.

[11] In terms of software, Bent and Caplan (2017) argue that the lattice allocation method can be implemented in small to medium-sized organizations using spreadsheet software such as Excel, and that large organizations can take advantage of specialized software where Bent and Caplan, among others, mention the NumPy library (and thus the Python programming language) as a possibility. An important advantage of using Python (and related programming languages) over spreadsheet software such as Excel is that all calculations, including the iterative optimization process in which the transformed matrices converge to a stable solution and final rules emerge, can be automated. Although it is not the objective of this paper, note that it also contributes to the literature by offering materials and codes to operationalize this relatively new method for allocating costs.

Table 2 presents the allocated costs and the final costs of the operating departments after the allocations, for each of the four methods (see Figures 1-4 for the associated output of the Python codes for the four methods). As shown in Table 2, the final costs of the operating departments after the allocations differ significantly between the different methods; however, the final costs based on the reciprocal and the lattice allocation methods are similar.[12] If we assume that the final costs based on these two methods are the most accurate, then usage of either the direct or step-down method leads to significant cross-subsidization between the operating departments and associated problems (such as perceptions of unfairness among their managers). Furthermore, notice that whereas based on the direct, step-down and reciprocal methods no costs are allocated from PD to VG and from IT to LT, this is different based on the lattice allocation method. This is due to the fact that the lattice allocation method also takes the support services provided by a support department indirectly to an operating department through another support department into account (Bent & Caplan, 2017). Also, notice that the lattice allocation method leaves some support department costs unallocated, which amounts become less with a lower cut-off value (i.e., with a higher desired degree of accuracy).

## The Python Codes and Related Exercises

The Togo (2012) case can easily be assessed using each of the four support department cost-allocation methods. When writing the Python codes (in version 3.7) for doing so, it was assumed that students have access to the Python ecosystem, in particular to the NumPy library, and preferably already have some experience with working with this library.[13] The following four exercises basically consist of writing or adapting Python codes to assess the case using each of the four methods.[14] When asking students to write and/or adapt the codes, it is recommended to provide them with the correct results (or even with the output of the Python code for the different methods), either ex ante or ex post, so that they can check whether they are doing (or have done) things correctly. All codes are structured in accordance with the case (i.e., reflecting a situation with four support departments and three operating departments); if one wants to use the methods for a situation with a different number of support and/or operating departments, the structure of the codes has to be adjusted accordingly. Also, all codes have been written in such a way that they can be used individually. If one uses the codes of multiple methods in combination, these can be shortened somewhat by making use of their overlap. Finally, it is important to emphasize that variations of these codes may work equally well and that students may thus come up with such variations.

Overall, the Python codes have a similar structure. All of them start with two NumPy arrays: (1) a one-dimensional array (i.e., a data structure to represent a vector of elements) in which the user has to input the department cost data, and (2) a two-dimensional array (i.e., a data structure to represent a matrix of elements) in which the user has to input the support services provided by the support departments to the other departments (expressed as proportions). In addition, (only) in the code for the lattice allocation method, the user also has to input the desired degree of accuracy (i.e., the cut-off value, which is now set to be 0.001). Then, in a number of steps that depend on the particular method (see below), the allocation percentages and allocated costs are calculated. Finally, the final costs of the operating departments after the allocations are calculated. This final part of the codes is also written to be the same for all methods.

The following exercises consist of writing Python codes that allocate the support department costs and calculate the final costs of the operating departments after the allocations using each of the four methods. See Section 5 for a discussion of alternative uses of these exercises and codes.

**Exercise 1: Write a Python code that allocates the support department costs using the direct method.** (See Appendix A for the Python code for this method.)

---

[12] Note that for the reciprocal method, the so-called gross services model (cf. Keller, 2015) is used.

[13] There are many excellent books and online sources on Python and the NumPy library available. Two highly recommended (free) online sources are the following: https://www.pythonlikeyoumeanit.com/ and https://lectures.quantecon.org/py/

[14] Similar Python codes for cases based on data from Bhimani et al. (2019), Togo (2010) and Bent and Caplan (2017) can be obtained from the author upon request.

Brief description: The essence of the direct method is that all costs of the support departments are directly allocated to the operating departments. To achieve this, first the numbers in the matrix (two-dimensional array) reflecting the support services provided are adjusted to reflect the allocation percentages. In the columns for the support departments, all numbers (except those on the diagonal, which must be equal to -1.00, as these reflect the services provided by a support department to the other departments) are set to 0.00. The numbers in the columns for the operating departments are recalculated by dividing for each support department, the proportion of support services provided to a certain operating department by the total proportion of support services provided by that support department to all operating departments combined. Here the Python code must include some auxiliary calculations (which for each support department calculate the sum of the proportions of support services provided by a support department to all operating departments combined) based on the initial matrix with support services provided. These are necessary because these numbers (and therefore their sum) change as part of the code, leading to erroneous outcomes if these auxiliary calculations are not included. Next, a matrix (two-dimensional array) reflecting the department costs is created, which is done by translating the initial vector (one-dimensional array) with cost data into a matrix. This matrix is structured in such a manner that the rows for the support departments consist of repeating series of references to the support departments' cost as included in the initial vector, while the rows for the operating departments consist of 0's except where it refers to its own department cost as included in the original vector. Finally, these two matrices are multiplied to calculate the allocated costs, after which the final costs of the operating departments are calculated. See Figure 1 for the output of the Python code for the direct method.
Possible hint(s): A possible hint that can be given to students when having them write the code from scratch, is to point out the importance of the auxiliary calculations.

**Exercise 2: Write a Python code that allocates the support department costs using the step-down method.** (See Appendix B for the Python code for this method.[15])
Brief description: The essence of the step-down method is that the costs of the support departments are allocated to other support departments and the operating departments sequentially in a stepwise fashion, where once the costs of a certain support department are allocated, none are allocated back in subsequent steps. To achieve this, first the numbers in the matrix (two-dimensional array) reflecting the support services provided are adjusted to reflect the allocation percentages. Similar to the direct method, in the columns for the support departments, the numbers on the diagonal must be equal to -1.00, as these reflect the services provided by a support department to the other departments. Different from the direct method, however, only the numbers below the diagonal in these columns are set to 0.00. The other numbers in these columns, as well as those in the columns for the operating departments, are recalculated (sequentially in a stepwise fashion and without allocating back to support departments earlier in the sequence) by dividing for each support department, the proportion of support services provided to a certain other support department (later in the sequence) or operating department by the total proportion of support services provided by that support department to all other support departments (later in the sequence) and operating departments combined. Similar to the direct method, here the Python code must include some auxiliary calculations (which for each support department calculate the sum of the proportions of support services provided by a support department to all other support departments (later in the sequence) and operating departments combined) based on the initial matrix with support services provided. These are necessary because these numbers (and therefore their sum) change as part of the code, leading to erroneous outcomes if these auxiliary calculations are not included. Next, a matrix (two-dimensional array) reflecting the department costs is created, which is done by translating the initial vector (one-dimensional array) with cost data into a matrix. This matrix is structured in such a manner that the rows for the support departments consist of repeating series of references to the support departments' cost as included in the initial vector, while the rows for the operating departments consist of 0's except where it refers to its own department cost as included in the original vector. Finally, these two matrices are multiplied to calculate the allocated costs, after which the final costs of the operating departments are calculated. See Figure 2 for the output of the Python code for the step-down method.

[15] This code assumes that the sequence of the departments has been determined ex ante and that the cost data and support services provided at the top of the code already reflect this sequence. It is also possible to write a code that also automates the sequencing. However, such a code would become much more lengthy and complicated, and is therefore less suitable for teaching the Python programming language to management accounting students. Following Togo (2012), the sequence in the case is based on the costs of the support departments. In general, given that the step-down method is the only method for which it matters, I advise to present the material to the students in the sequence necessary (based on the chosen criterion) for this method.

Possible hint(s): A possible hint that can be given to students when having them write the code from scratch, is to point out the importance of the auxiliary calculations.

**Exercise 3: Write a Python code that allocates the support department costs using the reciprocal method.** (See Appendix C for the Python code for this method.)
Brief description: The Python code for the reciprocal method is somewhat more complicated than those for the direct and step-down methods. The reciprocal method fully recognizes the interdepartmental services by allowing reallocations back to each support department, and uses matrix algebra operations for solving the simultaneous equations. More specifically, in order to calculate the allocated costs using this method, consistent with Togo's (2012) spreadsheet matrix approach, first the negative transpose matrix of the interdepartmental services is calculated. In other words, the rows/columns listing the proportions of services provided among the support departments are selected from the initial matrix (two-dimensional array) as a separate matrix, which then is transposed (i.e., flipped over its diagonal) and multiplied by -1.0 (cf. Togo, 2012). Then, a linear algebra function is used to calculate the reciprocated costs of the support departments and after that the allocated costs. Next, a matrix (two-dimensional array) reflecting the department costs is created, which is done by translating the initial vector (one-dimensional array) with cost data into a matrix. This matrix is structured in such a manner that the rows for the support departments consist of repeating series of references to the calculated reciprocated costs of the support departments, while the rows for the operating departments consist of 0's except where it refers to its own department cost as included in the original vector. Finally, these two matrices are multiplied to calculate the allocated costs, after which the final costs of the operating departments are calculated. See Figure 3 for the output of the Python code for the reciprocal method.
Possible hint(s): A possible hint that can be given to students when having them write the code from scratch, is to provide the NumPy function for the matrix algebra operations for solving the simultaneous equations (numpy.linalg.solve(a, b)).

**Exercise 4: Write a Python code that allocates the support department costs using the lattice allocation method.** (See Appendix D for the Python code for this method.)
Brief description: The Python code for the lattice allocation method is also somewhat more complicated than those for the direct and step-down methods. The essence of the lattice allocation method is that the so-called lattice allocation matrix (a matrix of fractions reflecting the percentages of service provisions among the different departments) is multiplied by itself, until the transformed matrix converges to a stable solution and final rules emerge, at which point the allocated costs approach those of the reciprocal method. To achieve this, first the lattice allocation matrix is created, which means that compared to the initial matrix (two-dimensional array) with support services provided, the -1.00 listed for support departments on the diagonal (representing their allocated reciprocal services) are replaced by 0.00. Then, the lattice allocation matrix is multiplied by itself until the cut-off value is reached, meaning that the desired accuracy is obtained. In the code, the percentages in the columns of the support departments are summed and these sums are compared with the cut-off value provided by the user. As long as one of these sums is larger than the cut-off value, the code iterates by taking the matrix to the next-highest power (i.e., by multiplying it with the (original) lattice matrix). Next, a matrix (two-dimensional array) reflecting the department costs is created, which is done by translating the initial vector (one-dimensional array) with cost data into a matrix. This matrix is structured in such a manner that the rows for the support departments consist of repeating series of references to the support departments' cost as included in the initial vector, while the rows for the operating departments consist of 0's except where it refers to its own department cost as included in the original vector. Finally, these two matrices are multiplied to calculate the allocated costs, after which the final costs of the operating departments are calculated. See Figure 4 for the output of the Python code for the lattice allocation method.
Possible hint(s): A possible hint that can be given to students when having them write the code from scratch, is to point out that a while-loop can be used for the matrix multiplication(s) needed to obtain the desired accuracy.

## How to Use the Case and the Python Codes

There are different ways in which the case and the Python codes presented in this paper can be used as teaching materials for classroom purposes. Depending on the current management accounting knowledge and

programming skills of the students, as well as the time available for solving and discussing the case, some important options are:[16]

1. To provide the students none of the codes and to have them do the calculations using the different methods by writing their own Python codes from scratch;
2. To provide the students part of the codes and to have them complete these Python codes to do the calculations using the different methods;
3. To provide the students all codes and to have them use these Python codes to do the calculations using the different methods for an adapted situation;
4. To provide the students all codes and to have them use these Python codes to do the calculations using the different methods.

The most informative and effective seem to be the two intermediate options. The existing case can easily be adapted by adapting the numbers (i.e., the department costs and/or the service percentages) and/or the structure (i.e., by adding or deleting one or more support and/or operating department(s)), where the number and difficulty of adjustments can be chosen to match the students' level. Students can also be asked to use only some of the four methods; for example, the direct and/or step-down method when students have only limited programming skills, and the reciprocal and/or lattice allocation method when students have more advanced programming skills. The main point is to have them actively work with the codes to do the calculations, and to use online resources (Google) to work through problems. This will not only increase their understanding of the different cost-allocation methods but simultaneously also enhance their programming skills.

It should be emphasized that, in particular when the case and the Python codes are used in a basic or intermediary management accounting course, these materials should be part of a much broader discussion of issues related to the allocation of costs of support departments (or cost allocation in general). The main strength of management accountants is typically not their programming skills, but their ability to interpret, understand and value the calculated numbers and their underlying assumptions, and to communicate about these numbers. Strengthening these abilities should therefore always be the main focus when discussing a case, irrespective of the tools used to do the calculations. [17] This is consistent with the professional accounting literature (see Dzuranin et al., 2018) and supported by the results of surveys. For example, in a recent survey among 267 accounting faculty, Dzuranin et al. (2018, p. 25) conclude that "when considering the specific data analytic skills and tools that are of primary importance to [the] accounting curriculum, faculty consider developing students' mindset as the most important." Similarly, in a recent survey among 342 accounting professionals, Brink and Stoel (2019, p. 25) find "stronger preferences for skills related to data interpretation and communication over any individual technical skill or statistical knowledge", and conclude that this suggests "a role for accountants as intermediaries who may need to translate data analytic activities into business language."

Of the different ways in which the case and the Python codes can be used as teaching materials, the most likely is the one in which either option 2 or 3 (see above) is followed and the materials are used in a basic or intermediary management accounting course. The learning objectives of the case are then: (1) to increase

[16] Alternatively, the Python codes can also be used by a lecturer to illustrate the different methods, as well as the conditions in which the four methods give different results, during class. As these conditions center around the issue of proportionality, the lecturer could then, for example, show the results of using the different methods when: (a) both the costs of the support departments and the usage of their services by the other departments are proportional, (b) only the costs of the support departments are disproportionate, (c) only the usage of their services by the other departments is disproportionate, and (d) both the costs of the support departments and the usage of their services by the other departments are disproportionate. In addition, the Python codes can of course also be used by a practitioner to analyze a real-life cost-allocation situation. Here it should be noted that for practitioners especially the Python codes for the reciprocal and lattice allocation methods are of interest, because the direct and step-down methods can also easily be done using software packages (such as Excel) that practitioners are typically already familiar with.

[17] In the context of the topic of this paper, some important issues that can be discussed, with more or less depth depending on the level of the course, are: (1) why it is important to do these calculations in the first place (i.e., why organizations allocate support department costs) (e.g., Kaplan & Atkinson, 1998); (2) how accurate the allocate costs, and as a consequence the final costs of the operating departments, are (e.g., Datar & Gupta, 1994); and (3) how important it is, depending on the specific circumstances of the organization, that these costs are accurate (e.g., Merchant & Shields, 2003).

students' understanding of the different cost-allocation methods (including the relatively new lattice allocation method), and (2) to enhance their Python programming skills. In terms of students' competences, the case mainly develops their technical skills (programming and calculation skills), but particularly if used as a homework assignment[18] that students have to hand in on paper and/or present to the other students, also their soft skills (written and/or verbal communication skills) are developed. In terms of students' programming skills, if either option 2 or 3 is followed, the case develops their ability to read, understand and adjust (i.e., complete or adapt) Python codes (which will also require their problem solving skills). The amount of time students need to solve the case ultimately depends on their current management accounting knowledge and programming skills in combination with the amount of code not provided to them (in case of option 2) or the number and difficulty of adjustments to the case (in case of option 3). The latter should be chosen to match the students' level, in such a way that it will take them about an hour to an hour and a half to solve the case. After the students have handed in their assignment, the case can then be discussed within the confines of a normal class setting, either with or without student presentations, but always as part of a much broader discussion of issues related to the allocation of costs of support departments (or cost allocation in general). Finally, it should be noted that while learning the basics of Python is not that difficult[19], teaching programming (whether in Python or another programming language) to students is sometimes considered to be challenging. Fortunately, there is a large and growing body of evidence about how this can best be done (e.g., Brown & Wilson, 2018; Luxton-Reilly et al., 2018; Porter et al., 2013).

## Summary and Conclusions

This paper explores how the Python programming language can be taught to management accounting students using domain-specific examples and exercises. Building on an existing case, the paper presents a number of Python codes that can be used as teaching materials in a management accounting course, and discusses how the case and the Python codes can be used in such a course. The materials cover a topic, support department cost-allocation methods, that is discussed in almost every management accounting course, and also include a relatively new approach known as the lattice allocation method. This topic was mainly chosen because the available methods for allocating the costs of support departments to other departments vary in terms of ease of use, which translates into Python codes that also vary in terms of difficulty and required functionalities.

It should be emphasized that it is not the objective of this paper, nor should it be the objective of any accounting curriculum, to try to turn accounting students into professional data scientists. After all, there are many specialized programs for this purpose. Given the big impact of the major new developments in ICT on the accounting profession, however, it is important that accounting students acquire adequate skills in data analytics, as well as (at least) a basic understanding of programming languages such as Python. Although for many accounting purposes software packages such as Excel can also be used, using Python (and related programming languages) has some clear advantages (such as greater flexibility, the ability to handle bigger datasets, and better reproducibility due to the fact that coding automatically provides an audit trail), which make it valuable for accounting students to acquire (at least) a basic understanding of such languages. As accounting faculty, we should therefore aim to provide them with enough knowledge and programming skills for them to: (1) understand the possibilities and limitations of data analytics, also to develop their ability to interpret, understand and value information that is based on data analytics methods such as machine learning, (2) learn new emerging technologies in the future, and (3) be able to communicate with specialized data scientists.

When teaching students how to write and/or adapt Python codes, it is most relevant and effective if this is done using domain-specific examples and exercises, because this will not only increase their understanding of the particular accounting topic(s) but simultaneously also enhance their programming skills. The materials presented in this paper, of which the efficacy may be empirically tested in future research, cover one specific management accounting topic, support department cost-allocation methods, but similar materials can easily be developed for other management accounting topics. For example, as Bent and Caplan (2017) illustrate that their lattice

[18] Alternatively, the case can also be used in a computer lab classroom, where students solve the case either individually or in pairs (see the tips for teaching programming of Brown & Wilson, 2018).

[19] There are many excellent books and online sources on the basics of Python available. Two highly recommended (free) online sources are the following: https://www.py4e.com/ and https://automatetheboringstuff.com/

allocation method can also be used to implement two-stage activity-based costing (ABC) and note that they believe their method is potentially useful for other types of complex allocations, these materials can easily be extended to such other types of cost allocations. More generally, given the quantitative nature of our methods and techniques, many other management accounting topics are equally suitable to be taught using programming languages such as Python. Overall, it is to be expected that more of such teaching materials will be developed in the near future, and I hope that this paper contributes to achieving this aim.

## References

AACSB International (2013). Eligibility procedures and accreditation standards for accounting accreditation. Available at: <http://www.aacsb.edu/accreditation/standards/2013-accounting>.

Ballou, B., Heitger, D. L., & Stoel, M. D. (2018). Data-driven decision-making and its impact on accounting undergraduate curriculum. *Journal of Accounting Education*, *44*, 14-24.

Bent, K., & Caplan, D. (2017). Lattice allocations: A better way to do cost allocations. *Advances in Accounting*, *38*, 99-105.

Bhimani, A., Horngren, C. T., Datar, S. M., & Rajan, M. V. (2019). *Management and Cost Accounting* (7th ed.). Harlow: Pearson.

Bransford, J., Sherwood, R., Vye, N., & Rieser, J. (1986). Teaching thinking and problem solving: Research foundations. *American Psychologist*, *41*(10), 1078-1089.

Brink, W. D., & Stoel, M. D. (2019). Analytics knowledge, skills, and abilities for accounting graduates. *Advances in Accounting Education: Teaching and Curriculum Innovations, 22*, 23-43.

Brown, N. C. C., & Wilson G. (2018). Ten quick tips for teaching programming. *PLoS Computational Biology, 14*(4): e1006023. Available at: <https://doi.org/10.1371/journal.pcbi.1006023>.

Christensen, D., & Schneider, P. (2017). Allocating service department costs with Excel. *Strategic Finance*, *98*(11), 50-55.

Datar, S., & Gupta, M. (1994). Aggregation, specification and measurement errors in product costing. *The Accounting Review, 69*(4), 567-591.

Drury, C. (2018). *Management and Cost Accounting* (10th ed.). Andover: Cengage.

Dzuranin, A. C., Jones, J. R., & Olvera, R. M. (2018). Infusing data analytics into the accounting curriculum: A framework and insights from faculty. *Journal of Accounting Education*, *43*, 24-39.

Fogarty, T. J. (2018). Forces of change – Another perspective: A reply to Pincus et al. (2017). *Journal of Accounting Education, 43*, 40-42.

Frey, C. B., & Osborne, M. A. (2017). The future of employment: How susceptible are jobs to computerisation? *Technological Forecasting and Social Change, 114*, 254-280.

Kaplan, R. S., & Atkinson, A. A. (1998). *Advanced Management Accounting* (3rd ed.). Upper Saddle River (N.J.): Prentice-Hall.

Keller, A. C. (2005). Simpler than ABC: New ideas for using Microsoft Excel for allocating costs. *Management Accounting Quarterly, 6*(4), 24-33.

Keller , A. C. (2015). Service department cost allocations using the net services model and the MDTERM function in Excel. *Journal of Accounting Education, 33*, 241-255.

Lawson, R. A. (2018). Management accounting education: New imperatives. *Strategic Finance, 100*(2), 40-45.

Lawson, R. A., & Smith, D. (2018). How to master digital age competencies. *Strategic Finance, 100*(3), 30-37.

Lucas Jr, H., Agarwal, R., Clemons, E. K., El Sawy, O. A., & Weber, B. (2013). Impactful research on transformational information technology: An opportunity to inform new audiences. *MIS Quarterly, 37*(2), 371-382.

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., & Szabo, C. (2018). Introductory programming: A systematic literature review. In: Proceedings of 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'18). ACM, New York (N.Y). Available at: <http://repository.falmouth.ac.uk/3051/1/ITiCSE_2018__WG3.pdf>.

Merchant, K. A., & Shields, M. D. (1993). When and why to measure costs less accurately to improve decision making. *Accounting Horizons, 7*(2), 76-81.

Pelzer, J. R. E., & DeLaurell, R. M. (2018). Implementation of AACSB Standard A7: A strategy for limited resources. *The Accounting Educators' Journal*, *28*, 117-138.

Pincus, K. V., Stout, D. E., Sorensen, J. E., Stocks, K. D., & Lawson, R. A. (2017). Forces for change in higher education and implications for the accounting academy. *Journal of Accounting Education, 40*, 1-18.

Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in introductory programming: What works? *Communications of the ACM*, *56*(8), 34-36.

Richins, G., Stapleton, A., Stratopoulos, T. C., & Wong, C. (2017). Big Data analytics: Opportunity or threat for the accounting profession? *Journal of Information Systems*, *31*(3), 63-79.

Seal, W., Rohde, C., Garrison, R. H., & Noreen, E. W. (2019). *Management Accounting* (6th ed.). London: McGraw-Hill Education.

Stinson, J. B. (2002). Cost allocation – From the simple to the sublime. *Management Accounting Quarterly, 4*(1), 1-10.

Togo, D. F. (2010). Maile-Ann Company: A matrix approach to reciprocated support department cost allocations. *Journal of Business Case Studies*, *6*(2), 35-40.

Togo, D. F. (2012). Support department cost allocations with a matrix-based reciprocal approach. *Advances in Accounting Education: Teaching and Curriculum Innovations, 13*, 277-296.

**Table 1: Cost Data and Support Services Provided**

| | Support departments | | | | Operating departments | | |
|---|---|---|---|---|---|---|---|
| | HR | PD | IT | MA | LT | VG | ER |
| Department costs: | 800,000 | 600,000 | 400,000 | 200,000 | 2,000,000 | 1,800,000 | 1,200,000 |
| Support by: | | | | | | | |
| Human resources (HR) | -1.00 | 0.15 | 0.25 | 0.10 | 0.40 | 0.05 | 0.05 |
| Product development (PD) | 0.25 | -1.00 | 0.20 | 0.15 | 0.35 | 0.00 | 0.05 |
| Information technology (IT) | 0.20 | 0.10 | -1.00 | 0.05 | 0.00 | 0.45 | 0.20 |
| Maintenance (MA) | 0.10 | 0.05 | 0.15 | -1.00 | 0.05 | 0.35 | 0.30 |
| Laptop computers (LT) | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 |
| Video games (VG) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 |
| Early readers (ER) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |

**Table 2: Allocated and Final Costs According to the Different Methods**

| Department costs | Support departments | | | | Operating departments | | |
|---|---|---|---|---|---|---|---|
| | HR | PD | IT | MA | LT | VG | ER |
| | 800,000 | 600,000 | 400,000 | 200,000 | 2,000,000 | 1,800,000 | 1,200,000 |
| Direct method | | | | | | | |
| HR | -800,000 | 0 | 0 | 0 | 640,000 | 80,000 | 80,000 |
| PD | 0 | -600,000 | 0 | 0 | 525,000 | 0 | 75,000 |
| IT | 0 | 0 | -400,000 | 0 | 0 | 276,923 | 123,076 |
| MA | 0 | 0 | 0 | -200,000 | 14,285 | 100,000 | 85,714 |
| Final costs | 0 | 0 | 0 | 0 | 3,179,285 | 2,256,923 | 1,563,791 |
| Step-down method | | | | | | | |
| HR | -800,000 | 120,000 | 200,000 | 80,000 | 320,000 | 40,000 | 40,000 |
| PD | 0 | -720,000 | 192,000 | 144,000 | 335,999 | 0 | 48,000 |
| IT | 0 | 0 | -792,000 | 56,571 | 0 | 509,142 | 226,285 |
| MA | 0 | 0 | 0 | -480,571 | 34,326 | 240,285 | 205,959 |
| Final costs | 0 | 0 | 0 | 0 | 2,690,326 | 2,589,428 | 1,720,244 |
| Reciprocal method | | | | | | | |
| HR | -1,275,935 | 191,390 | 318,983 | 127,593 | 510,374 | 63,796 | 63,796 |
| PD | 228,746 | -914,985 | 182,997 | 137,247 | 320,244 | 0 | 45,749 |
| IT | 195,809 | 97,904 | -979,049 | 48,952 | 0 | 440,572 | 195,809 |
| MA | 51,379 | 25,689 | 77,069 | -513,793 | 25,689 | 179,827 | 154,138 |
| Final costs | 0 | 0 | 0 | 0 | 2,856,308 | 2,484,197 | 1,659,494 |
| Lattice allocation method | | | | | | | |
| HR | 152 | 101 | 165 | 88 | 431,420 | 218,248 | 149,822 |
| PD | 133 | 87 | 142 | 76 | 325,255 | 149,724 | 124,579 |
| IT | 61 | 40 | 65 | 35 | 67,913 | 221,214 | 110,669 |
| MA | 24 | 16 | 26 | 14 | 31,283 | 94,524 | 74,110 |
| Final costs | 372 | 244 | 399 | 215 | 2,855,873 | 2,483,711 | 1,659,183 |

**Figure 1: Output of the Python Code for the Direct Method**

```
Department costs:
[ 800000  600000  400000  200000 2000000 1800000 1200000]

Allocation percentages:
[[-1.    0.    0.    0.    0.8   0.1   0.1 ]
 [ 0.   -1.    0.    0.    0.88  0.    0.13]
 [ 0.    0.   -1.    0.    0.    0.69  0.31]
 [ 0.    0.    0.   -1.    0.07  0.5   0.43]
 [ 0.    0.    0.    0.    1.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    1.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    1.  ]]

Cost allocation (based on the direct method):
[[-800000       0       0       0  640000   80000   80000]
 [      0 -600000       0       0  525000       0   75000]
 [      0       0 -400000       0       0  276923  123076]
 [      0       0       0 -200000   14285  100000   85714]
 [      0       0       0       0 2000000       0       0]
 [      0       0       0       0       0 1800000       0]
 [      0       0       0       0       0       0 1200000]]

Final costs of department LT: $3179285
Final costs of department VG: $2256923
Final costs of department ER: $1563791

Total costs (grand total): $7000000
```

**Figure 2: Output of the Python Code for the Step-Down Method**

```
Department costs:
[ 800000  600000  400000  200000 2000000 1800000 1200000]

Allocation percentages:
[[-1.    0.15  0.25  0.1   0.4   0.05  0.05]
 [ 0.   -1.    0.27  0.2   0.47  0.    0.07]
 [ 0.    0.   -1.    0.07  0.    0.64  0.29]
 [ 0.    0.    0.   -1.    0.07  0.5   0.43]
 [ 0.    0.    0.    0.    1.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    1.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    1.  ]]

Cost allocation (based on the step-down method):
[[-800000  120000  200000   80000  320000   40000   40000]
 [      0 -720000  192000  144000  335999       0   48000]
 [      0       0 -792000   56571       0  509142  226285]
 [      0       0       0 -480571   34326  240285  205959]
 [      0       0       0       0 2000000       0       0]
 [      0       0       0       0       0 1800000       0]
 [      0       0       0       0       0       0 1200000]]

Final costs of department LT: $2690326
Final costs of department VG: $2589428
Final costs of department ER: $1720244

Total costs (grand total): $6999999
```

**Figure 3: Output of the Python Code for the Reciprocal Method**

```
Department costs:
[ 800000  600000  400000  200000 2000000 1800000 1200000]

Negative transpose matrix:
[[ 1.   -0.25 -0.2  -0.1 ]
 [-0.15  1.   -0.1  -0.05]
 [-0.25 -0.2   1.   -0.15]
 [-0.1  -0.15 -0.05  1.  ]]

Reciprocated costs of the support departments:
[1275935  914985  979049  513793]

Cost allocation (based on the reciprocal method):
[[-1275935   191390   318983   127593   510374    63796    63796]
 [  228746  -914985   182997   137247   320244        0    45749]
 [  195809    97904  -979049    48952        0   440572   195809]
 [   51379    25689    77069  -513793    25689   179827   154138]
 [       0        0        0        0  2000000        0        0]
 [       0        0        0        0        0  1800000        0]
 [       0        0        0        0        0        0  1200000]]

Final costs of department LT: $2856308
Final costs of department VG: $2484197
Final costs of department ER: $1659494

Total costs (grand total): $7000000
```

**Figure 4: Output of the Python Code for the Lattice Allocation Method**

```
Department costs:
[ 800000  600000  400000  200000 2000000 1800000 1200000]

Lattice allocation matrix:
[[0.   0.15 0.25 0.1  0.4  0.05 0.05]
 [0.25 0.   0.2  0.15 0.35 0.   0.05]
 [0.2  0.1  0.   0.05 0.   0.45 0.2 ]
 [0.1  0.05 0.15 0.   0.05 0.35 0.3 ]
 [0.   0.   0.   0.   1.   0.   0.  ]
 [0.   0.   0.   0.   0.   1.   0.  ]
 [0.   0.   0.   0.   0.   0.   1.  ]]

Number of matrix multiplications needed to obtain the desired accuracy
: 9

Lattice allocation matrix at the desired accuracy level:
[[0.   0.   0.   0.   0.54 0.27 0.19]
 [0.   0.   0.   0.   0.54 0.25 0.21]
 [0.   0.   0.   0.   0.17 0.55 0.28]
 [0.   0.   0.   0.   0.16 0.47 0.37]
 [0.   0.   0.   0.   1.   0.   0.  ]
 [0.   0.   0.   0.   0.   1.   0.  ]
 [0.   0.   0.   0.   0.   0.   1.  ]]

Cost allocation (based on the lattice allocation method):
[[    152     101     165      88  431420  218248  149822]
 [    133      87     142      76  325255  149724  124579]
 [     61      40      65      35   67913  221214  110669]
 [     24      16      26      14   31283   94524   74110]
 [      0       0       0       0 2000000       0       0]
 [      0       0       0       0       0 1800000       0]
 [      0       0       0       0       0       0 1200000]]

Final costs of department LT: $2855873
Final costs of department VG: $2483711
Final costs of department ER: $1659183

Total costs (grand total): $6998768
```

## Appendix A: Python Code for the Direct Method

```
# Direct method
# Illustration based on data taken from Togo (2012)
# Case with 4 support departments (HR, PD, IT and MA) and 3 operating departments (LT, VG, ER)

import numpy as np

# Cost data and support services provided

dep_costs = np.array([800000,600000,400000,200000,2000000,1800000,1200000])
serv_perc = np.array([[-1.00,0.15,0.25,0.10,0.40,0.05,0.05],
                      [0.25,-1.00,0.20,0.15,0.35,0.00,0.05],
                      [0.20,0.10,-1.00,0.05,0.00,0.45,0.20],
                      [0.10,0.05,0.15,-1.00,0.05,0.35,0.30],
                      [0.00,0.00,0.00,0.00,1.00,0.00,0.00],
                      [0.00,0.00,0.00,0.00,0.00,1.00,0.00],
                      [0.00,0.00,0.00,0.00,0.00,0.00,1.00]])

print("Department costs:")
print(dep_costs)

# Auxiliary calculations

aux_a = serv_perc[0, 4:].sum()
aux_b = serv_perc[1, 4:].sum()
aux_c = serv_perc[2, 4:].sum()
aux_d = serv_perc[3, 4:].sum()

# Calculate the allocation percentages

serv_perc[:,0] = 0.00
serv_perc[0,0] = -1.00
serv_perc[:,1] = 0.00
serv_perc[1,1] = -1.00
serv_perc[:,2] = 0.00
serv_perc[2,2] = -1.00
serv_perc[:,3] = 0.00
serv_perc[3,3] = -1.00
serv_perc[0,4] = (serv_perc[0,4] / aux_a)
serv_perc[0,5] = (serv_perc[0,5] / aux_a)
serv_perc[0,6] = (serv_perc[0,6] / aux_a)
serv_perc[1,4] = (serv_perc[1,4] / aux_b)
serv_perc[1,5] = (serv_perc[1,5] / aux_b)
serv_perc[1,6] = (serv_perc[1,6] / aux_b)
serv_perc[2,4] = (serv_perc[2,4] / aux_c)
serv_perc[2,5] = (serv_perc[2,5] / aux_c)
serv_perc[2,6] = (serv_perc[2,6] / aux_c)
serv_perc[3,4] = (serv_perc[3,4] / aux_d)
serv_perc[3,5] = (serv_perc[3,5] / aux_d)
serv_perc[3,6] = (serv_perc[3,6] / aux_d)
print("\nAllocation percentages:")
print(serv_perc.round(2))

# Calculate the allocated costs

D = np.array([[dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0]],
```

```
        [dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1]],
        [dep_costs[2],dep_costs[2],dep_costs[2],dep_costs[2],dep_costs[2],dep_costs[2],dep_costs[2]],
        [dep_costs[3],dep_costs[3],dep_costs[3],dep_costs[3],dep_costs[3],dep_costs[3],dep_costs[3]],
        [0,0,0,0,dep_costs[4],0,0],
        [0,0,0,0,0,dep_costs[5],0],
        [0,0,0,0,0,0,dep_costs[6]]])

all_costs = np.multiply(serv_perc, D)
print("\nCost allocation (based on the direct method):")
print(all_costs.astype(int))

# Calculate the final costs of the operating departments after the allocations

col5_sum = all_costs[:, 4].sum()
col6_sum = all_costs[:, 5].sum()
col7_sum = all_costs[:, 6].sum()
print("\nFinal costs of department LT: $" + str(int(col5_sum)))
print("Final costs of department VG: $" + str(int(col6_sum)))
print("Final costs of department ER: $" + str(int(col7_sum)))
print("\nTotal costs (grand total): $" + str(int((col5_sum + col6_sum + col7_sum))))
```

## Appendix B: Python Code for the Step-Down Method

```
# Step-down method
# Illustration based on data taken from Togo (2012)
# Case with 4 support departments (HR, PD, IT and MA) and 3 operating departments (LT, VG, ER)

import numpy as np

# Cost data and support services provided

dep_costs = np.array([800000,600000,400000,200000,2000000,1800000,1200000])
serv_perc = np.array([[-1.00,0.15,0.25,0.10,0.40,0.05,0.05],
                      [0.25,-1.00,0.20,0.15,0.35,0.00,0.05],
                      [0.20,0.10,-1.00,0.05,0.00,0.45,0.20],
                      [0.10,0.05,0.15,-1.00,0.05,0.35,0.30],
                      [0.00,0.00,0.00,0.00,1.00,0.00,0.00],
                      [0.00,0.00,0.00,0.00,0.00,1.00,0.00],
                      [0.00,0.00,0.00,0.00,0.00,0.00,1.00]])

print("Department costs:")
print(dep_costs)

# Calculate the allocation percentages

serv_perc[1:,0] = 0.00
serv_perc[2:,1] = 0.00
serv_perc[3:,2] = 0.00
serv_perc[4:,3] = 0.00

aux_a = serv_perc[1, 2:].sum()

serv_perc[1,2] = (serv_perc[1,2] / aux_a)
serv_perc[1,3] = (serv_perc[1,3] / aux_a)
serv_perc[1,4] = (serv_perc[1,4] / aux_a)
serv_perc[1,5] = (serv_perc[1,5] / aux_a)
serv_perc[1,6] = (serv_perc[1,6] / aux_a)

aux_b = serv_perc[2, 3:].sum()

serv_perc[2,3] = (serv_perc[2,3] / aux_b)
serv_perc[2,4] = (serv_perc[2,4] / aux_b)
serv_perc[2,5] = (serv_perc[2,5] / aux_b)
serv_perc[2,6] = (serv_perc[2,6] / aux_b)

aux_c = serv_perc[3, 4:].sum()

serv_perc[3,4] = (serv_perc[3,4] / aux_c)
serv_perc[3,5] = (serv_perc[3,5] / aux_c)
serv_perc[3,6] = (serv_perc[3,6] / aux_c)

dep_costs[1] = dep_costs[1] + serv_perc[0,1] * dep_costs[0]
dep_costs[2] = dep_costs[2] + serv_perc[0,2] * dep_costs[0] + serv_perc[1,2] * dep_costs[1]
dep_costs[3] = dep_costs[3] + serv_perc[0,3] * dep_costs[0] + serv_perc[1,3] * dep_costs[1] + serv_perc[2,3] *
dep_costs[2]

print("\nAllocation percentages:")
print(serv_perc.round(2))
```

```
# Calculate the allocated costs

D = np.array([[dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0]],
              [dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1]],
              [dep_costs[2],dep_costs[2],dep_costs[2],dep_costs[2],dep_costs[2],dep_costs[2],dep_costs[2]],
              [dep_costs[3],dep_costs[3],dep_costs[3],dep_costs[3],dep_costs[3],dep_costs[3],dep_costs[3]],
              [0,0,0,0,dep_costs[4],0,0],
              [0,0,0,0,0,dep_costs[5],0],
              [0,0,0,0,0,0,dep_costs[6]]])

all_costs = np.multiply(serv_perc, D)
print("\nCost allocation (based on the step-down method):")
print(all_costs.astype(int))

# Calculate the final costs of the operating departments after the allocations

col5_sum = all_costs[:, 4].sum()
col6_sum = all_costs[:, 5].sum()
col7_sum = all_costs[:, 6].sum()
print("\nFinal costs of department LT: $" + str(int(col5_sum)))
print("Final costs of department VG: $" + str(int(col6_sum)))
print("Final costs of department ER: $" + str(int(col7_sum)))
print("\nTotal costs (grand total): $" + str(int((col5_sum + col6_sum + col7_sum))))
```

## Appendix C: Python Code for the Reciprocal Method

```
# Reciprocal method
# Illustration based on data taken from Togo (2012)
# Case with 4 support departments (HR, PD, IT and MA) and 3 operating departments (LT, VG, ER)

import numpy as np

# Cost data and support services provided

dep_costs = np.array([800000,600000,400000,200000,2000000,1800000,1200000])
serv_perc = np.array([[-1.00,0.15,0.25,0.10,0.40,0.05,0.05],
                      [0.25,-1.00,0.20,0.15,0.35,0.00,0.05],
                      [0.20,0.10,-1.00,0.05,0.00,0.45,0.20],
                      [0.10,0.05,0.15,-1.00,0.05,0.35,0.30],
                      [0.00,0.00,0.00,0.00,1.00,0.00,0.00],
                      [0.00,0.00,0.00,0.00,0.00,1.00,0.00],
                      [0.00,0.00,0.00,0.00,0.00,0.00,1.00]])

print("Department costs:")
print(dep_costs)

# Calculate the negative transpose matrix of the interdepartmental services

A = (serv_perc[0:4, 0:4].transpose()) * - 1.0
print("\nNegative transpose matrix:")
print(A)

# Calculate the allocated costs

B = np.array([dep_costs[0],dep_costs[1],dep_costs[2],dep_costs[3]])
C = np.linalg.solve(A, B)
print("\nReciprocated costs of the support departments:")
print(C.astype(int))

D = np.array([[C[0],C[0],C[0],C[0],C[0],C[0],C[0]],
              [C[1],C[1],C[1],C[1],C[1],C[1],C[1]],
              [C[2],C[2],C[2],C[2],C[2],C[2],C[2]],
              [C[3],C[3],C[3],C[3],C[3],C[3],C[3]],
              [0,0,0,0,dep_costs[4],0,0],
              [0,0,0,0,0,dep_costs[5],0],
              [0,0,0,0,0,0,dep_costs[6]]])

all_costs = np.multiply(serv_perc, D)
print("\nCost allocation (based on the reciprocal method):")
print(all_costs.astype(int))

# Calculate the final costs of the operating departments after the allocations

col5_sum = all_costs[:, 4].sum()
col6_sum = all_costs[:, 5].sum()
col7_sum = all_costs[:, 6].sum()
print("\nFinal costs of department LT: $" + str(int(col5_sum)))
print("Final costs of department VG: $" + str(int(col6_sum)))
print("Final costs of department ER: $" + str(int(col7_sum)))
print("\nTotal costs (grand total): $" + str(int((col5_sum + col6_sum + col7_sum))))
```

**Appendix D: Python Code for the Lattice Allocation Method**

```
# Lattice allocation method
# Illustration based on data taken from Togo (2012)
# Case with 4 support departments (HR, PD, IT and MA) and 3 operating departments (LT, VG, ER)

import numpy as np

# Cost data, support services provided and desired accuracy

dep_costs = np.array([800000,600000,400000,200000,2000000,1800000,1200000])
serv_perc = np.array([[-1.00,0.15,0.25,0.10,0.40,0.05,0.05],
                      [0.25,-1.00,0.20,0.15,0.35,0.00,0.05],
                      [0.20,0.10,-1.00,0.05,0.00,0.45,0.20],
                      [0.10,0.05,0.15,-1.00,0.05,0.35,0.30],
                      [0.00,0.00,0.00,0.00,1.00,0.00,0.00],
                      [0.00,0.00,0.00,0.00,0.00,1.00,0.00],
                      [0.00,0.00,0.00,0.00,0.00,0.00,1.00]])
cut_off = 0.001

print("Department costs:")
print(dep_costs)

# Create the lattice allocation matrix

serv_perc[0,0] = 0.00
serv_perc[1,1] = 0.00
serv_perc[2,2] = 0.00
serv_perc[3,3] = 0.00
print("\nLattice allocation matrix:")
print(serv_perc)

# Matrix multiplication(s) needed to obtain the desired accuracy

col1_sum = serv_perc[:, 0].sum()
col2_sum = serv_perc[:, 1].sum()
col3_sum = serv_perc[:, 2].sum()
col4_sum = serv_perc[:, 3].sum()
n = 0
while (col1_sum > cut_off) | (col2_sum > cut_off) | (col3_sum > cut_off) | (col4_sum > cut_off):
   n += 1
   serv_perc_adj = np.linalg.matrix_power(serv_perc, n)
   col1_sum = serv_perc_adj[:, 0].sum()
   col2_sum = serv_perc_adj[:, 1].sum()
   col3_sum = serv_perc_adj[:, 2].sum()
   col4_sum = serv_perc_adj[:, 3].sum()
print("\nNumber of matrix multiplications needed to obtain the desired accuracy: " + str(n))
print("\nLattice allocation matrix at the desired accuracy level:")
print(serv_perc_adj.round(2))

# Calculate the allocated costs

D = np.array([[dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0],dep_costs[0]],
          [dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1],dep_costs[1]],
          [dep_costs[2],dep_costs[2],dep_costs[2],dep_costs[2],dep_costs[2],dep_costs[2],dep_costs[2]],
          [dep_costs[3],dep_costs[3],dep_costs[3],dep_costs[3],dep_costs[3],dep_costs[3],dep_costs[3]],
          [0,0,0,0,dep_costs[4],0,0],
```

```
            [0,0,0,0,0,dep_costs[5],0],
            [0,0,0,0,0,0,dep_costs[6]]])

all_costs = np.multiply(serv_perc_adj, D)
print("\nCost allocation (based on the lattice allocation method):")
print(all_costs.astype(int))

# Calculate the final costs of the operating departments after the allocations

col5_sum = all_costs[:, 4].sum()
col6_sum = all_costs[:, 5].sum()
col7_sum = all_costs[:, 6].sum()
print("\nFinal costs of department LT: $" + str(int(col5_sum)))
print("Final costs of department VG: $" + str(int(col6_sum)))
print("Final costs of department ER: $" + str(int(col7_sum)))
print("\nTotal costs (grand total): $" + str(int((col5_sum + col6_sum + col7_sum))))
```